The Automated Generation of Test Cases using an Extended Domain Based Reliability Model

Alberto Avritzer
Siemens Corporate Research
755 College Road East
Princeton, NJ 08540
alberto.avritzer@siemens.com

Elaine J. Weyuker
AT&T Labs - Research
180 Park Avenue
Florham Park, NJ 07932
weyuker@research.att.com

Abstract

We present a new approach for the automated generation of test cases to be used for demonstrating the reliability of large industrial mission-critical systems. In this paper we extend earlier work by adding failure tracking and transient Markov chain analysis. Results from the transient Markov chain analysis are used to estimate the software reliability at a given system execution time.

1 Introduction

We introduce a new model-based test case generation approach to be used to assess the reliability of large industrial mission-critical systems. For such systems, the reliability objective is usually specified in terms of the probability of failure-free operation for a certain period of time under given operational conditions. Therefore, to study the reliability evolution of a large industrial system, the reliability model must track failure evolution as a function of the system execution time, t, under realistic operational conditions.

With this goal in mind, we extend the software reliability model introduced in [3] by adding both failure tracking and system execution time. Both the reliability modeling and test case generation approaches that we will present in this paper use the function $p(n, t)$, the transient value of the probability of program P's correct execution, for input n and time t.

In our earlier paper [3] we introduced an automated approach for test case generation and test case execution that was applied to several large industrial telecommunications systems. These systems were modeled by using Markov chains because the arrival processes could be modeled as Poisson processes, and it was reasonable to assume that the service times were exponentially distributed. We further assumed that the types of reliable telecommunications systems we were studying were designed to operate at low to medium utilization rates. Again, this was a realistic assumption for such systems.

Therefore, the test case generation approaches we introduced, known as *Deterministic State Testing* or *DST*, were able to achieve the specified model coverage by using search algorithms based on the underlying spanning-tree generated by the depth first search algorithms rooted at the idle state. In that approach, all states with steady-state probability values that were determined to be smaller than an empirically-defined ϵ could be safely discarded because of the low to medium utilization assumption.

Taking the DST approach as a starting point, we now extend the test case generation algorithm to incorporate both the resource usage perspective used in [3], and also resource failures.

The outline of the remainder of the paper is as follows. In Section 2 we survey the relevant literature with particular emphasis on Markov modeling approaches that are useful for our automated test case generation and reliability estimation approaches. In Section 3 we extend the DST test case generation algorithm introduced in [3] to use the transient solution of the Markov model generated by the Tangram-II tool. Section 4 describes how we have extended the reliability estimation approach introduced in [3] by including system execution time. In Section 5 we present some empirical results, while Section 6 contains our conclusions and suggestions for future research.

2 Related Work

In [11], Whittaker and Thomason introduced a statistical testing approach supported by Markov models representing usage behavior. In this model, the Markovian state definition captures information related to the user interface navigation. For a given state i, the transition probability to

state j, denoted $p_{i,j}$, represents the probability of navigating from state i to state j. The Markov chain in this approach is a discrete-time Markov chain, with no loops. The authors propose using complete navigation patterns to select test cases, since the whole path probability can be computed as the product of its individual transition probabilities.

Building on this approach, Kallepalli and Tian [9] presented a method for Web testing and reliability analysis supported by Web log data. The authors used a Unified Markov Model to support usage-based statistical testing.

In [13], Yan et al. introduced a scenario usage diagram model that could be derived from annotated UML model diagrams. Their approach was also based on the Whittaker and Thomason Markov chain usage model, using a method for acceleration of statistical testing.

Woit introduced a tool for automated test case generation and reliability estimation [12]. This tool was based on random sampling of a module's expected inputs, with the reliability estimation based on module testing results.

Cortelessa et al. presented a component-based reliability estimation approach to be applied early in the project's life cycle [8]. In their proposed methodology, a Bayesian framework for reliability prediction was created, based on reliability-annotated UML diagrams.

Markov chain models can also be used to model system failures and to support reliability computation [5]. Avritzer et al. used a discrete-state continuous-time Markov chain. The Markovian state, $S(t)$, contained a list of resources that had failed up to time t. These Markov chains were used to provide a reliability estimation for a given system execution time t. Defining reliability to be the probability of failure-free operation at time t, it was computed by evaluating the transient Markov chain states' probability distribution for time t, and then summing over the set of states that represented failure-free operation. They used the Tangram-II [10] tool transient solution method which provides different solution techniques for both the stationary and transient analysis of the Markovian model.

This approach provided a very practical means of estimating reliability as a function of system execution time and expected failure rates, since the Markovian state was defined as the number of failed resources at time t. Therefore, the Markovian state explosion problem could be solved by limiting attention to only those states that have a certain number of failed resources. The drawback of this approach is that resource usage is not modeled.

For this reason, the new approach introduced in this paper combines the DST perspective [3] which models resource usage but does not include reliability computation based on system execution time and failures, with the system execution time-based approach described in [5].

Our approach is also related to the ones introduced in [8] and [12], which use a Bayesian framework to evaluate the overall system reliability by decomposing the system into modules. In contrast, the algorithm which we will introduce in Section 3, decomposes the system Markov chain state into a resource usage part and a failure part. Pass/fail criteria is assessed for each resource usage test case, for all possible failure states. We then apply a Bayesian framework to compute the overall system reliability.

3 The Test Case Generation Algorithm

We now introduce our automated test case generation approach that takes into account both resource usage and resource failure events. The intuition is that when designing test suites for performance testing, we want to select test cases that occur frequently because the impact of a failure in such a state will be significant since it happens often in practice. In addition, we also want to select test cases that have a high probability of failure, even if the likelihood of occurrence is relatively low. We determine likelihood of failure based on past behavior.

By using decomposition theory, we generate two separate Markov chains: a resource usage-based Markov chain and a resource failure-based Markov chain. We use the Deterministic State Testing approach introduced in [3] for automated test case generation.

In the first type of Markov chain, the number of failure events of each type are explicitly tracked. In the second type of Markov chain, the state definition does not explicitly track failures; it is instead defined in terms of the number of resources of each type currently in use and resource allocation and deallocation are explicitly tracked.

By combining the two types of Markov chains, we are able to focus on both types of issues - testing the inputs that are most likely to occur, and testing the inputs that are most likely to fail. In both cases these classes are determined based on historical data.

We begin by assuming that resources may be allocated and deallocated both in the course of normal system operation and also due to failure and repair events. We apply decomposition and aggregation concepts [7] to generate the two Markov chains.

The first step involves applying our original DST algorithm to generate a failure-based Markov chain. For this Markov chain, a state $F1$ represents the number of resources of each type that have been consumed by a failure event. Assuming that there are N distinct types of resources, we represent a state by an N-tuple $(r_1, r_2, ..., r_N)$ with r_i denoting the number of resources of type i that are *unavailable* because of failures. The state $(0, 0, ..., 0)$ for this Markov chain is known as the *ok state* because no resources have failed.

After building this failure-based Markov chain, we build a second resource usage-based Markov chain. This is done

by applying the DST algorithm to a state definition $R1$ that represents the number of resources of each type that have been consumed by a resource allocation event in the course of normal system operation. Again the states are N-tuples, $(r_1, r_2, ..., r_N)$, but for this Markov chain, r_i denotes the number of resources of type i that are being used at the current time. The state $(0, 0, ..., 0)$ for this Markov chain is known as the *idle state* because no resources are being used.

For the Markov chain defined by state $F1$, resource failure events are assumed to form a Poisson process with average failure rates denoted λ_f. Resource repair events are assumed to be exponentially distributed for this Markov chain with average μ_f, where f represents the index to the failed resource.

The homogeneous Poisson assumption has typically been made in the reliability modeling literature when the number of faults in a system is constant once the system has been deployed, and we also make this assumption.

In addition, the average time for repair is generally a reasonable approximation because the maintenance workforce is often composed of a fixed number of people, and it is assumed that each person performs work with a service time that is approximately exponentially distributed.

For the Markov chain defined by state $R1$, resource allocation events are assumed to form a Poisson process with average rates λ_r and resource deallocation events are assumed to be exponentially distributed with average μ_r, where r represents an index to the allocated/deallocated resource.

The Markov chain state $F1$ for the failure-based model is defined to be the number of resources of each type that have been consumed by a failure event. Let x represent a generic system-wide resource, and let

$$\lambda_x = \lambda_f \tag{1}$$

Similarly, for resource repair we have,

$$\mu_x = \mu_f \tag{2}$$

Therefore, letting N_x be the number of resources of type x, the ratio $\frac{\lambda_x}{N_x \mu_x}$ represents the average utilization of resource x, or the probability that the resource x is found to be busy, for the resource usage-based Markov chain. When $\frac{\lambda_x}{\mu_x}$ ¡ 1, then resource requests are coming in at a slower rate than they are being completed, so the resource is unlikely to be busy, in contrast to when $\frac{\lambda_x}{\mu_x} \geq 1$.

Similarly, for the failure-based Markov chain, the ratio $\frac{\lambda_x}{N_x \mu_x}$ represents the probability of finding resource x in the failed state.

Strategy: Generate all states that are likely to occur in practice as represented by having a steady-state probability of occurrence greater than ϵ, as well as those states that are likely to be busy, or in a failed state and use this as the basis for test case generation.

Algorithm : Generate a list of test cases starting from the software state S.

1. Index x denotes the resource type. Set x to 1.

2. If by adding one more resource of type x, a previously unreached state is reached and the steady-state probability of the state so generated is greater than ϵ or $\frac{\lambda_x}{\mu_x} \geq 1$ then:

   - Generate a test case for the software state reached from S by adding one more resource failure of type x. Call it state S';
   - Generate a list of test cases by recursively executing this algorithm on (S');
   - If x is greater than N, the number of resource types, return;
   - Otherwise set the index of resource types to $x+1$, and go to step 2.

To generate the Markov chain associated with resource allocation/deallocation events, we use the same algorithm with the appropriate change of state definition and resource allocation/deallocation rates.

For the second model, recall that the Markov chain state $R1$ is defined as the number of resources of each type that have been consumed by a resource allocation event. Let x represent a generic system-wide resource, and let,

$$\lambda_x = \lambda_r \tag{3}$$

Similarly, for resource deallocation we have,

$$\mu_x = \mu_r \tag{4}$$

This algorithm can be used to automatically generate test cases when the objective is to estimate reliability as a function of system execution time, for a specific operational environment. The Markov chain generated by the execution of the algorithm on (F1) is solved using transient analysis methods. Each state generated by the execution of the algorithm on (R1) is weighted by the transient failure probability associated with it and by the pass/fail assessment of the state.

The set of states that have a steady state probability greater than ϵ for the Markov chain associated with resource allocation/deallocation events defines a test suite that may have thousands of test cases. This test suite can be efficiently executed by automatically driving the system to the resource configuration specified in each test case.

4 Reliability Modeling and Assessment Approach

We have introduced a new approach to the generation of the most likely states in the Markov chain, taking into account both resource usage and resource failures. In contrast, the original DST algorithm introduced in [3] only took resource usage into account. The reliability assessment approach introduced in that paper was domain-based and did not have any provision for system execution time. The equation below describes the domain-based reliability assessment approach introduced in [3]:

$$\alpha(n) = \begin{cases} 0 & \text{if } P(n) = S(n) \\ 1 & \text{otherwise.} \end{cases}$$

Then $\sum_{n \epsilon D} p(n)\alpha(n)$ is the probability that a run of program P with input n chosen according to the probability distribution p will result in a failure. D represents the input domain, n is an element in the input domain and P represents a program that was designed to implement specification S. It then follows that

$$R(P) = 1 - \sum_{n \epsilon D} p(n)\alpha(n) = \sum_{n \epsilon D} p(n)(1 - \alpha(n))$$

is the probability that a run with input n chosen according to p will result in a correct execution. Here correctness does not refer to functional correctness, but rather to a lack of resource exhaustion.

In this section, we extend this reliability assessment approach by including system execution time. We use the new algorithm introduced in Section 3 to generate the most likely states in the two Markov chains. We then use the transient analysis method to generate the state probability distribution for the required system execution time, using the state space derived by the algorithm for the failure-based Markov chain.

The summation of the state probabilities for the states generated by the algorithm that represent failure-free operation provide an accurate estimation of the system reliability. A state is defined as being failure-free if resource allocation is successful when the system is in that state. In contrast, a failed state represents the condition in which a resource allocation cannot be satisfied because of a resource failure that occurred before the system entered that state.

The process for reliability modeling and reliability assessment using domain-based reliability modeling is as follows:

1. Define system state and required input parameters,
2. Run the algorithm to generate a test suite,
3. Execute the generated test suite to obtain pass/fail conditions for each test case,
4. Use the domain-based reliability model with steady-state probability distribution $p(n)$ for reliability assessment.

In the domain-based reliability modeling approach, we let the set of states that have been tested be denoted by $1, 2, \ldots, z$, with s_s^i denoting the intended state corresponding to the i^{th} test case. Then the software reliability after a deterministic state testing algorithm application is given by:

$$R[p, S](P) = 1 - \sum_{i=1}^{z} p(i)\alpha(i) \tag{5}$$

where,

$$\alpha(i) = \begin{cases} 0 & \text{if } P(i) = s_s^i \\ 1 & \text{otherwise.} \end{cases}$$

The process for reliability modeling using system execution time-based modeling is as follows:

1. Define failure-based system state $F1$ and execute the algorithm to generate a test suite T_{F1},
2. Use a Markov chain solver to obtain the transient solution of the Markov chain, for the specified system execution time,
3. Define resource usage-based system state $R1$ and execute the algorithm to generate a test suite T_{R1},
4. For each state $R1S$ in the test suite T_{R1}, evaluate pass/fail condition for the state $R1S$ for each state in test suite T_{F1}.
5. To obtain the reliability assessment for the system, we compute the weighted sum of the state probabilities that are failure-free. This step consists of the application of Bayes Theorem.

The transient solution of the failure-based Markov chain for a given system execution time, creates a probability distribution that is used to compute the overall probability for each state of the resource usage Markov chain. In Equation 6, the term $p_t(j)$ is obtained from the transient solution of the failure-based Markov chain for a given system execution time t. $\alpha(i, j)$ is the pass/fail assessment for the resource usage test case i for the failure-based Markov chain state j. $p(i|j)$ is the steady state probability of the resource usage Markov chain state i conditioned on the occurrence of the resource failure represented by failure state j.

In the system execution time-based reliability modeling approach, we let the set of failure-based states that have been tested be denoted by $1, 2, \ldots, y$. We let the set of resource usage-based states that have been tested be denoted by $1, 2, \ldots, z$. We let s_{ij} denote the intended state corresponding to the i^{th} resource usage-based test case, when resource failure j is active.

$$R[p,S,t](P) = 1 - \sum_{i=1}^{z}\sum_{j=1}^{y} p(i|j)p_t(j)\alpha(i,j) \quad (6)$$

where,

$$\alpha(i,j) = \begin{cases} 0 & \text{if } P(i,j) = s_{ij} \\ 1 & \text{otherwise.} \end{cases}$$

In our approach, both resource usage-based and failure-based test cases are derived from the associated Markov chains. Therefore, we use i to denote both test cases and their associated Markov chain states. When we apply the decomposition assumption to the solutions of the failure-based Markov chain and the resource usage-based Markov chain, $p(i|j) = p(i)$, as the solution of the resource usage-based Markov chain is independent of the solution of the failure-based Markov chain. Therefore, we can rewrite Equation 6 as Equation 7 and the software reliability, for a given execution time t, after a deterministic state testing algorithm application is given by:

$$R[p,S,t](P) = 1 - \sum_{i=1}^{z} p(i)\sum_{j=1}^{y} p_t(j)\alpha(i,j) \quad (7)$$

5 Empirical Results

In this section we apply our new algorithm to generate test cases and assess the reliability of two systems which are similar to actual large industrial systems that we have worked with. Recalling that the state $(0,0,...0)$ is called the *ok state* because no resources have failed, the first example represents a system with five operations and a steady state probability of being in the ok state of 0.869. The second example represents a system with three operations and a steady state probability of being in the ok state of 0.995.

5.1 Example 1

Table 1 contains the resource allocation and deallocation rates, while Table 2 contains the estimated failure and repair rates for this sample system.

Failure rates for the most critical function of the example is assumed to be 1 failure per 1000 hours and for the other functions, the failure rate is assumed to be 5 failures per 1000 hours. We assume the maintenance process is unified and most functions will be repaired in 8 hours, except that the most critical function will get highest priority and so be repaired in 4 hours. When we applied our algorithm with the data shown in Table 2, the following test suite characteristics for the failure-based Markov chain were observed:

1. *ok state probability* is 0.869
2. Total probability mass covered is 1.0
3. Number of test cases needed for probability mass coverage of 1.0 is 2,002
4. Number of test cases needed for probability mass coverage of 0.99999 is 39
5. Number of test cases needed for probability mass coverage of 0.9999 is 26
6. Number of test cases needed for probability mass coverage of 0.999 is 14
7. Number of test cases needed for probability mass coverage of 0.99 is 5

For the 0.99 probability mass coverage the test case configuration and the test case probabilities for the 5 test cases are shown in Table 3.

For the 0.999 probability mass coverage, the test case configuration and the test case probabilities for the 9 additional test cases required to cover 0.999 are shown in Table 4.

Note that the first operation has a lower failure rate and a higher repair rate and therefore a lower probability of occurrence. Therefore, the failure modes investigated do not include the failure of the first operation as shown in Table 3 and Table 4.

We are now ready to determine test cases based solely on the operational usage of the system. This is known as the resource usage Markov chain, and it is derived from the resource allocation and deallocation rates contained in Table 1. The total number of test cases that were generated by our algorithm, for the resource usage Markov chain test suite, based on the selected value of ϵ and the target probability mass coverage, was 24,492 and the total probability mass coverage associated with these 24,492 test cases was 0.964. The probability mass associated with the idle state $p_0 = (0,0,0,0,0)$, was 0.00000005. For Example 1, the five most likely to occur resource configurations for the resource usage Markov chain are shown in Table 5, and would therefore be selected as test cases.

We now show the application of Equation 7 to Example 1 for the most probable resource usage test case, $(9,2,2,1,0)$, for which the resource usage-based Markov chain steady state probability is 0.02. We evaluate Equation 7 for the 0.99 reliability objective case, shown in Table 3, which has five failure-based states.

We assume for illustration purposes, that test case $(9,2,2,1,0)$, passed for resource failure conditions $(0,0,0,0,0)$, $(0,0,0,0,1)$, and $(0,0,0,1,0)$. That means we assume that the resource allocation requests for the

Resource Type	Avg Arr Rate (calls/min)	Avg Hldg Time (Minutes)
OP1	3.3	3.0
OP2	0.823	3.0
OP3	0.588	2.0
OP4	0.297	1.0
OP5	0.588	5.0

Table 1. Example 1: Resource Allocation and Deallocation Rates

Resource Type	Failures/hour	Average Repair Time (Hours)
OP1	0.001	4.0
OP2	0.005	8.0
OP3	0.005	8.0
OP4	0.005	8.0
OP5	0.005	8.0

Table 2. Example 1: Resource Failure and Repair Rates

N1	N2	N3	N4	N5	Prob
0	0	0	0	0	0.869
0	0	0	0	1	0.035
0	0	0	1	0	0.035
0	0	1	0	0	0.035
0	1	0	0	0	0.017

Table 3. Example 1: Test Suite for 0.99 Probability Mass Coverage based on Resource Failure and Repair Rates

N1	N2	N3	N4	N5	Prob
0	0	0	1	1	0.0014
0	0	1	0	1	0.0014
0	0	1	1	0	0.0014
0	0	0	0	2	0.0007
0	0	0	2	0	0.0007
0	0	2	0	0	0.0007
0	1	0	0	1	0.0007
0	1	0	1	0	0.0007
0	1	1	0	0	0.0007

Table 4. Example 1: Test Suite for 0.999 Probability Mass Coverage based on Resource Failure and Repair Rates

resource usage test case $(9,2,2,1,0)$, were correctly executed for these resource failure conditions. Therefore $\alpha((9,2,2,1,0),j)$ is set to 0 for these states.

In addition, we assume that test case $(9,2,2,1,0)$ failed for the resource failure conditions $(0,0,1,0,0)$ and $(0,1,0,0,0)$. Therefore, we assume that the resource allocation requests for the resource usage test case $(9,2,2,1,0)$ were not correctly executed for these resource failure conditions.

It then follows that $\alpha((9,2,2,1,0),j)$ is set to 1 for these states. The application of Equation 7 for the state $(9,2,2,1,0)$, for time $t = 0$, for Example 1 is shown in Equation 8.

$$R[p,(9,2,2,1,0),0](P) = 1-0.02\times(0.035+0.017) \quad (8)$$

Therefore, instead of deducting the total state probability for state $(9,2,2,1,0)$, we instead only deduct the cases under which the test case failed. Assuming that this test case failed only under the failure conditions $(0,0,1,0,0)$ and $(0,1,0,0,0)$, we deduct $(0.035+0.017)\times 0.02 = 0.00104$. The reliability due to the observed failures for test case $(9,2,2,1,0)$ is therefore estimated from Equation 8 to be 1 - .00104 = 0.99896.

5.2 Example 2

Table 6 contains the resource allocation and deallocation rates, while Table 7 contains the estimated failure and repair rates for the second example.

The failure rate for the most critical function of the second system is assumed to be 0.1 failure per 1000 hours and for the other functions the rate is assumed to be 1 failure per 1000 hours. We assume the maintenance process is unified and most functions will be repaired in 4 hours, although again it is assumed that the most critical function will get highest priority and so be repaired in 1 hour.

When we applied our algorithm with the data shown in Table 7, the following test suite characteristics for the failure-based Markov chain were observed:

1. *ok state probability* is 0.995
2. Total probability mass covered is 1.0
3. Number of test cases needed for probability mass coverage of 1.0 is 220
4. Number of test cases needed for probability mass coverage of 0.99999 is 4
5. Number of test cases needed for probability mass coverage of 0.9999 is 3
6. Number of test cases needed for probability mass coverage of 0.999 is 3
7. Number of test cases needed for probability mass coverage of 0.99 is 1

The most likely test cases and their associated test case probability are shown in Table 8. For the total probability coverage of 0.999 and 0.9999 the three most likely test cases needed to be exercised.

The resource usage Markov chain for the second system was derived from the resource allocation and deallocation rates contained Table 6. The total number of test cases in the resource usage Markov chain test suite was 1,275 and the total probability mass coverage associated with these 1,275 test cases was 1.0. The probability mass associated with the idle state $p_0 = (0,0,0)$, was 0.18. For Example 2, the five most probable test cases for the resource usage Markov chain are shown in Table 9.

We now show the application of Equation 7 to Example 2 for the most probable resource usage test case, $(0,1,0)$, for which the resource usage-based Markov chain steady state probability is 0.2. We evaluate Equation 7 for the 0.99999 reliability objective case, shown in Table 8, which has four failure-based states. We assume that test case $(0,1,0)$ passed for resource failure conditions $(0,0,0)$, $(0,0,1)$, and $(0,1,0)$. Therefore, we assume that the resource allocation requests for the resource usage test case $(0,1,0)$, were correctly executed for these resource failure conditions. It then follows that $\alpha((0,1,0),j)$ is set to 0 for these states.

In addition, we assume that test case $(0,1,0)$, failed for resource failure condition $(0,0,2)$. Therefore, we assume that the resource allocation requests for the resource usage test case $(0,1,0)$ were not correctly executed for this resource failure condition. Therefore $\alpha((0,1,0),j)$ is set to 1 for this state. The application of Equation 7 for the state $(0,1,0)$, for time $t = 0$, for Example 2 is shown in Equation 9.

$$R[p,(0,1,0),0](P) = 1 - 0.2 \times 0.00000796 \quad (9)$$

Therefore, we observe that instead of deducting the total state probability of state $(0,1,0)$ due to the observed failure, $(0.00000796) \times 0.2 = 0.000001592$ is deducted. It then follows that the reliability due to the observed failures for test case $(0,1,0)$ is estimated from Equation 8 to be 0.999998408.

6 Conclusions

In this paper we have extended our automated test case generation approach introduced in [3] to define a test suite that includes both the most likely to occur inputs, and also

N1	N2	N3	N4	N5	Prob
9	2	2	1	0	0.002
10	2	2	1	0	0.00199
9	3	2	1	0	0.00197
10	3	2	1	0	0.00195
8	2	2	1	0	0.00183

Table 5. Example 1: Five Most Probable Test Cases based on Resource Allocation and Resource Deallocation rates

Resource Type	Avg Arr Rate (calls/min)	Avg Hldg Time (Minutes)
OP1	0.01667	30.0
OP2	0.037	10.0
OP3	0.06	30.0

Table 6. Example 2: Resource Allocation and Deallocation Rates

Resource Type	Failures/hour	Average Repair Time (Hours)
OP1	0.0001	1.0
OP2	0.001	4.0
OP3	0.001	4.0

Table 7. Example 2: Resource Failure and Repair Rates

N1	N2	N3	Prob
0	0	0	0.995
0	0	1	0.00398
0	1	0	0.000995
0	0	2	0.00000796

Table 8. Example 2: Most Likely Probability Mass Coverage based on Resource Failure and Repair Rates

N1	N2	N3	Prob
0	1	0	0.201
0	1	1	0.120
0	2	0	0.111
0	0	1	0.109
0	2	1	0.067

Table 9. Example 2: Five Most Probable Test Cases based on Resource Allocation and Resource Deallocation rates

the most likely to fail inputs. Our new algorithm takes advantage of Markov chain decomposition theory [7] to decompose the state space into two Markov chains that are modeled and solved independently.

We have applied our concepts to two sample systems, which are similar to large industrial systems we have worked with [3]. We have demonstrated that our algorithms are able to reduce the required modeling effort significantly according to the target reliability objective, provided the assumptions made are realistic.

For the first example, the number of test cases generated for the failure-based Markov chain was reduced from 2,002 for the reliability objective of 1.0 to 5 for the reliability objective of 0.99. For the reliability objective of 0.99999 only 39 test cases were required. For the second example, the number of test cases generated for the failure-based Markov chain was reduced from 220 for the reliability objective of 1.0 to 1 for the reliability objective of 0.99. For the reliability objective of 0.99999 only 4 test cases were required.

We are currently designing an approach to certify a very large mission-critical system for reliability and we are planning to use the reliability estimation algorithm introduced in this paper to generate the resource usage test cases and to derive the plots of the reliability metric as a function of system execution time. This will represent a real usage of the approach described in this paper. It will allow us to collect empirical data to verify the Poisson distribution for failure events and the exponential distribution for failure repairs, and to evaluate the impact of these assumptions on the size and complexity of our automatically generated test suites.

References

[1] A. Avizienis and D. E. Ball. On the achievement of a highly dependable and fault-tolerant air traffic control system. *IEEE Computer*, 1987, pp 84-90.

[2] A. Avritzer and B. Larson. Load testing software using deterministic state testing. In T. Ostrand and E.J.Weyuker, editors, *Proc. of the International Symposium on Software Testing and Analysis(ISSTA)*. ACM Press, June 1993, pp. 82-88.

[3] A. Avritzer and E. J. Weyuker. The Automatic Generation of Load Test Suites and the Assessment of the Resulting Software. *IEEE Trans. on Software Engineering*, Sept 1995, pp. 705-716.

[4] A. Avritzer, J. Ros and E. J. Weyuker. Reliability Testing of Rule-Based Systems. *IEEE Software*, September 1996, pp. 76–82.

[5] A. Avritzer, F. P. Duarte, R. M. M. Leo, E. S. Silva, M. Cohen, D. Costello. Reliability Estimation for Large Distributed Software Systems. *Cascon*. October 2008.

[6] M. O. Ball. Computational Complexity of Network Reliability Analysis: An Overview. *IEEE Trans. on Reliability*, Vol. R-35, NO.3, August, 1986, pp. 230-238.

[7] P. J. Courtois and P. Semal. Bounds for the Positive Eigenvectors of Nonnegative Matrices and for their Approximations by Decomposition. *J. ACM*, vol. 31, no. 4, pp. 804–824, 1984.

[8] V. Cortelessa, H. Singh, and B. Cukic. Early Reliability Assessment of UML based Software Models. *Proc. of the Third International Workshop on Software and Performance*, WOSP 2002, Rome, Italy, 2002, pp 302-309.

[9] C. Kallepalli and J. Tian. Measuring and Modeling Usage and Reliability for Statistical Web Testing. *IEEE Transactions on Software Engineering*, Vol. 27, No. 11, November 2001.

[10] E. de Souza e Silva, R. M. M. Leão, Richard R. Muntz, Ana P. C. da Silva, Antonio A. de A. Rocha, Flávio P. Duarte, Fernando J. S. Filho, Guilherme D. G. Jaime, Modeling, analysis, measurement and experimentation with the Tangram-II integrated environment. In *Proc. of Int. Conf. on Performance Evaluation Methodologies and Tools (ValueTools'06)*, 2006.

[11] J. A. Whittaker and M. G. Thomason. A Markov Chain Model for Statistical Software Testing. *IEEE Trans. Software Eng., vol. 20, no. 10, pp 812-824*, Oct. 1994.

[12] D. M. Woit. A Framework for Reliability Estimation. *Proceedings of the 5th International Symposium on Software Reliability Engineering*. Nov. 1994, pp. 18 - 24.

[13] J. Yan, J. Wang, and H-C. Chen. UML Based Statistical Testing Acceleration of Distributed Safety Critical Software. *Proceedings of the Second International Symposium on Parallel and Distributed Processing and Applications*, Hong Kong, China, Dec. 2004, Springer.